Multicast Tree Aggregation in Large Domains

Joanna Moulierac[1⋆], Alexandre Guitton[2⋆⋆], and Miklós Molnár[3⋆⋆⋆]

[1] IRISA/University of Rennes I, 35042 Rennes, France
[2] Birkbeck College, University of London, England
[3] IRISA/INSA, Rennes, France

Abstract. Tree aggregation is an efficient proposition that can solve the problem of multicast forwarding state scalability. The main idea of tree aggregation is to force several groups to share the same delivery tree: in this way, the number of multicast forwarding states per router is reduced. Unfortunately, when achieving tree aggregation in large domains, few groups share the same tree and the aggregation ratio is small. In this paper, we propose a new algorithm called TALD (Tree Aggregation in Large Domains) that achieves tree aggregation in domains with a large number of nodes. The principle of TALD is to divide the domain into several sub-domains and to achieve the aggregation in each of the sub-domain separately. In this way, there is possible aggregation in each of the sub-domain and the number of forwarding states is significantly reduced. We show the performance of our algorithm by simulations on a Rocketfuel network of 200 routers.

Keywords. Multicasting, tree aggregation, network simulation.

1 Introduction

With the growth of the number of network applications, it has been found a few years ago that the bandwidth was a bottleneck. Multicast has been developed to spare the bandwidth by sending efficiently copies of a message to several destinations. Although many research has been done on multicast, its deployment on the Internet is still an issue. This is due mainly to the large number of multicast forwarding states and to the control explosion when there are several concurrent multicast groups. Indeed, in the current multicast model, the number of multicast forwarding states is proportional to the number of multicast groups. The number of multicast groups is expected to grow tremendously together with the number of forwarding states: this will slow down the routing and saturate the routers memory. Additionally, the number of control messages required to maintain the forwarding states will grow in the same manner. This scalability issue has to be solved before multicast can be deployed over the Internet.

⋆ joanna.moulierac@irisa.fr
⋆⋆ alexandre@dcs.bbk.ac.uk
⋆⋆⋆ miklos.molnar@irisa.fr

Tree aggregation is a recent proposition that greatly reduces both the number of multicast forwarding states and the number of control messages required to maintain them. To achieve this reduction, tree aggregation forces several groups to share the same multicast tree. In this way, the number of multicast forwarding states depends on the number of trees and not on the number of groups.

1.1 Tree aggregation

The performance of tree aggregation mechanisms depends on how different groups are aggregated to the same tree within a domain. To aggregate several groups to the same tree, a label corresponding to the tree is assigned to all the multicast packets at the ingress routers of the domain. In the domain, the packets are forwarded according to this label. The label is removed at the egress routers so that the packets can be forwarded outside the domain. In addition to the multicast forwarding states that allow to match an incoming label to a set of outgoing interfaces, the border routers of the domain have to store group-specific entries. A group-specific entry matches a multicast address with a label.

Let us show the tree aggregation mechanism on an example. Figure 1 represents a domain with four border routers and the group-label table of the border router b_1. The two groups g_1 and g_3 can be aggregated to the same tree corresponding to label l_1 while g_2 uses its own label l_2. If a new group g_4 has members attached to routers b_1 and b_4, the tree manager can also aggregate g_4 to label l_1 or to the label l_2. In this case, no new tree is built but bandwidth is wasted with l_1 (resp. with l_2) when the messages for g_4 reach b_2 (resp. reach b_3) unnecessarily. Otherwise, the tree manager can build a new tree with label l_3 for g_4. In this case, no bandwidth is wasted but more forwarding states are required. Therefore, there is a trade-off between the wasted bandwidth and the number of states.

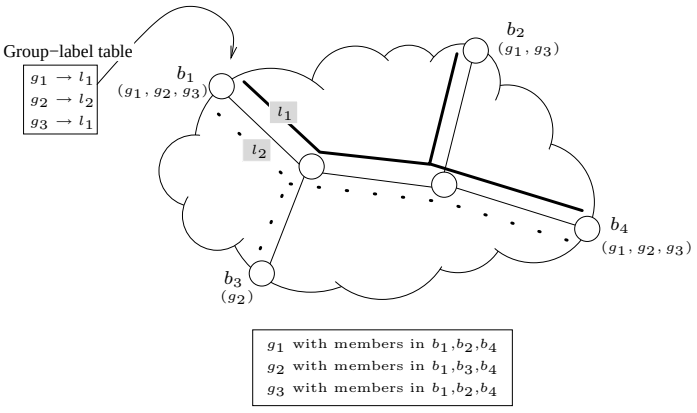

Fig. 1. Tree aggregation in a small domain.

1.2 Limits of tree aggregation in large domains

With tree aggregation, the number of forwarding states is proportional to the number of trees and not to the number of groups as in traditional multicast. However, when the domain is too large, we show that tree aggregation builds as many trees as traditional multicast and that there is no reduction of the number of forwarding states. Indeed, the number of different groups increase with the number b of border routers in the domain and the number g of concurrent groups. Therefore, it can be identified as the expected number of non-empty urns obtained by randomly throwing g balls into 2^b urns :

$$\textit{Number of different groups} = 2^b(1 - (1 - 2^{-b})^g)$$

This formula gives the total number of different groups in a domain with b border routers when there are g concurrent groups. We assume that the size of each group is chosen uniformly and that the members of each groups are chosen uniformly. This assumptions correspond to the worst-case scenario, where there is no correlation between groups.

Consequently, when there are too many border routers, the number of different groups is too large and the probability of finding a tree already existing for a new group is low. We will show in this paper that the existing protocols achieve tree aggregation within small domains of around 20 to 40 border routers but perform few aggregations in larger domain. Consequently, in large domains, the number of forwarding states is not reduced compared to traditional multicast and a new protocol has to be proposed in order to manager the large domains.

1.3 Proposition : sub-domain tree aggregation

In this paper, we propose a new protocol that performs aggregations in large domains. This protocol, TALD, for Tree Aggregation in Large Domains, divides the network into several sub-domains before aggregating. In this way, aggregation is feasible in each of the sub-domain and the number of forwarding states is strongly reduced.

Let us suppose that the domain is divided into d domains of approximately the same number of nodes. The union of the d domains is equal to D and the domains are disjoint. Thus, the number of different groups can be seen as :

$$\textit{Number of different groups} = 2^{b/d}(1 - (1 - 2^{-b/d})^g) \times d$$

For example, on a network with 15 border routers, there are 8 618 different groups for 10 000 concurrent groups using the formula above. However, on a network with 40 border routers, there are 10 000 different groups for 10 000 concurrent groups. Consequently, if the members of the groups are distributed uniformly, there are not two group with exactly the same members for 10 000 concurrent groups. Now, if the domain is divided into 4 sub-domains with approximately 10 nodes, the total number of different groups is equal to 4000 (there

are approximately 1000 different groups for 10 000 concurrent groups for a domain with 10 border routers). Consequently, when the domain is divided into several sub-domains, the number of different groups decreases significantly.

The rest of the paper is organized as follows. Section 2 describes an algorithm that divides the domain into several sub-domains and describes the aggregation protocol TALD. Section 3 validates the algorithm by showing its performance on simulations. Section 4 describes the existing protocols for tree aggregation. Section 5 concludes and gives the perspectives of our work.

2 The protocol TALD

In this section, we show how to design the protocol TALD (Tree Aggregation in Large Domains) that achieves sub-domains tree aggregation. Three main issues arise in order to present TALD:

1. How to divide the domains into several sub-domains?
2. How to aggregate groups within a sub-domain?
3. How to route packets in the domain for the multicast group, considering the aggregation of the sub-domains?

2.1 Dividing a domain into sub-domains

In order to minimize the total number of different multicast groups, the domain D has to be divided into sub-domains D_i of approximately the same number of nodes. We propose an algorithm that divides the domain $D = (V, E)$ into two sub-domains $D_1 = (V_1, E_1)$ and $D_2 = (V_2, E_2)$ where $V_i \subset V$ is the set of routers of the domain D_i and $E_i \subset E$ the set of links.

The main idea of the algorithm is to find first the two nodes x_1 and x_2 with the maximum distance in the domain D, *i.e.* the two most distant nodes. Then, two sets of nodes V_1 and V_2 are created with $x_1 \in V_1$ and $x_2 \in V_2$. Iteratively, the nearest nodes of the nodes already in the set are added; at each step of the algorithm one node is added in V_1 and one node is added is V_2. When all the nodes of the domain D are whether in V_1 or in V_2, two domains $D_1 = (V_1, E_1)$ and $D_2 = (V_2, E_2)$ are built from the two sets. The edges in E_i are the edges including in E connecting two nodes in V_i. When the two sub-domains have been built, this algorithm can be reapplied on each of the sub-domain in order to get 4 sub-domains or more.

Figure 2 shows the network Eurorings[4] divided into four separated sub-domains by the algorithm presented in this subsection. The network was divided into two sub-domains and then they were also divided into two in order to obtain four separated sub-domains with disjoint sets of nodes of approximately the same size.

[4] `http://www.cybergeography.org/atlas/kpnqwest_large.jpg`

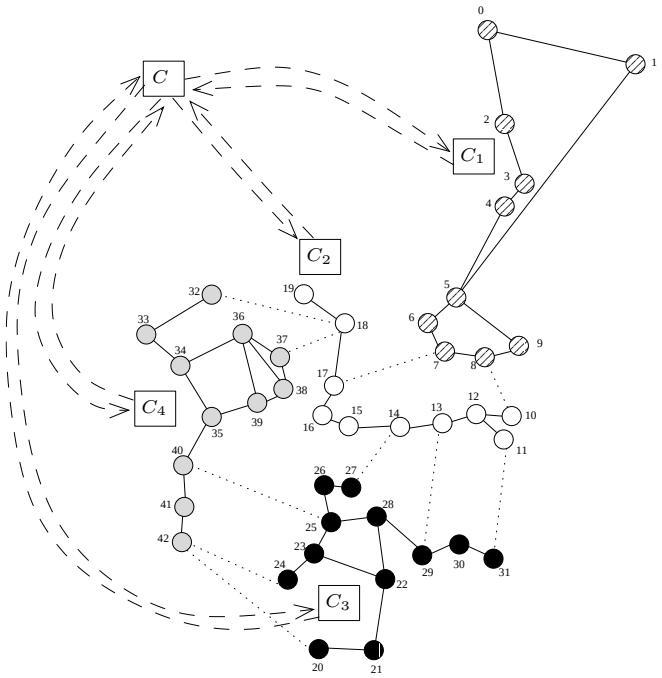

Fig. 2. Eurorings network divided into four sub-domains

2.2 Aggregating in a sub-domain

We assume in this subsection that the domain is divided into sub-domains. If the domain is already explicitly divided into sub-domains (e.g. for administrative reasons for example), there is no need to apply the algorithm described in the previous subsection.

Each sub-domain $D_i = (V_i, E_i)$ is controlled by a centralized entity C_i which is in charge of aggregating the groups within the sub-domain. For example, in figure 2, the sub-domain 1 is controlled by C_1. Each C_i knows the topology of the sub-domain (in order to build trees for the multicast groups) and maintains the group memberships for its sub-domain. Note that C_i is aware of only the members in its sub-domains and not the members for all the group.

When a border router receives a `join` or `leave` message for a group g, it forwards it to the centralized entity C_i of its sub-domain in its sub-domain. Then, C_i creates or updates the group specific entries for g in order to route the messages. The centralized entity C_i builds a native tree t_i covering the routers attached to members of g in its sub-domain, and then C_i tries to find an existing tree t_i^{agg} already configured in its sub-domain satisfying these two conditions:

– t_i^{agg} covers all the routers of the sub-domain attached to members of g

– the cost of t_i^{agg} (*i.e.* the sum of the cost of each link of t_i^{agg}) is not more than b_t% of the cost of the native tree t_i where b_t is a given bandwidth threshold:

$$cost(t_i^{agg}) \leq cost(t_i) \times (1 + b_t)$$

The centralized entity C_i chooses among all the trees matching these two conditions the tree t_i^{agg} with minimum cost. Then g is aggregated to t_i^{agg} and C_i updates in all the border routers attached to members of g a group specific entry matching g to t_i^{agg} (or more precisely, matching g to the label corresponding to t_i^{agg}: $g \rightarrow \text{label}(t_i^{agg})$). If no tree satisfies these two conditions, then C_i configures t_i (the tree initially built for g) by adding forwarding states in all the routers covered by t_i and then adds the group specific entries $g \rightarrow \text{label}(t_i)$.

2.3 Routing in the whole domain

The centralized entities C_i having members of g in their sub-domains have use the algorithm described in previous subsection. In order to route packets for the whole multicast group, the trees in all the sub-domains have to be connected. The centralized entity C, responsible of the main domain D is in charge of this task. Note that C does not need to know the topology of D to connect these trees. Several solutions are possible to connect these trees. We present in this paper a simple solution to connect these trees in order to validate first the main idea of our algorithm.

In this simple solution, each C_i, having members of g in its sub-domain i, has communicated to C the IP address of one of the routers of the sub-domain attached to members of g. This router is the representative router for g in D_i. The centralized entity C keeps this information and maintains the list of the representatives of g for each sub-domain. Note that C does not keep any information concerning the group memberships. Then, C connect the trees in the sub-domains by adding tunnels. The tunnels can be built by adding group specific entries matching g to routers in the others sub-domains.

For example in figure 2, suppose that C receives a message (g,`@IP(router 5)`) from C_1, a message (g,`@IP(router 11)`) from C_2 and a message (g,`@IP(router 28)`) from C_3. In this example, C has to connect the three trees corresponding to group g in the three sub-domains. In order to achieve this connection, C adds a group specific entry $g \rightarrow$`@IP(router 11)` in router 5. Two more are added in router 11: $g \rightarrow$`@IP(router 5)` and $g \rightarrow$`@IP(router 28)` and one in router 28: $g \rightarrow$`@IP(router 11)`. In this way, the three trees in the three sub-domains are connected by tunnels and messages for g can be routed.

As our concerns in this paper is to reduce the number of entries stored, we do not optimize the connection of the trees. This can be done as further part of investigation. What only matters for the moment is the number of group specific entries added. If three C_i have registered members of g to C, four group specific entries are added. More generally, if n C_i have replied to C, then $2(n-1)$ entries are needed.

3 Simulations

We run several simulations on different topologies. Due to lack of space, we present only the results of the simulations on the Rocketfuel graph Exodus [5]. This network contains 201 routers and 434 links. During the simulations, 101 routers were core routers and 100 others routers were border routers and can be attached to members of multicast groups. The plots are the results of 100 cases of simulations where each case corresponds to a different set of border routers.

We present the results of the protocols TALD-1, TALD-2 and TALD-4 for different bandwidth thresholds: when 0% bandwidth is allowed to be wasted and when 20% of bandwidth wasted. The protocol TALD-1 represents the actual tree aggregation protocols when the domain is not divided and when aggregation is performed in the main domain. With TALD-2, the domain in divided into 2 sub-domains and with TALD-4, the domain is divided into 4 sub-domains. The division was performed by the algorithm presented in Section 2.1.

The number of multicast concurrent groups varied from 1 to 10 000 and the number of members of groups was randomly chosen between 2 and 20. The members of groups were chosen randomly among the 100 border routers. This behavior is not representative of the reality but it allows to show the performance of the algorithms in worst-case simulation. Indeed, when the members are randomly located, then the aggregation is more difficult than if members of groups are chosen with some affinity model.

3.1 Number of forwarding states

Figure 3 plots the total number of forwarding states in the domain, *i.e.* the sum of the forwarding states stored by all the routers of the domain. Recall that for a bidirectional tree t, $|t|$ forwarding states have to be stored where $|t|$ denotes the number of routers covered by t. With TALD-1, there is almost no aggregation (the number of forwarding states is the same as if no aggregation was performed) and then, the number of multicast forwarding states is the same with 0% and with 20% of bandwidth wasted. The protocol TALD-4 gives significantly better results than TALD-1 and TALD-2. Moreover, with TALD-4, the number of multicast forwarding states is reduced when the bandwidth threshold is equal to 20%.

For example, TALD-4 stores around 160 000 forwarding states in the whole domain when the bandwidth threshold is equal to 0% for 10 000 concurrent groups. There is a reduction of 22% when the bandwidth threshold is equal to 20%: the number of forwarding states reaches approximately 126 000. Oppositely, the amount of bandwidth wasted has no influence for the results of TALD-1 as the number of forwarding states is the same when 0% of bandwidth is wasted and when 20% of bandwidth is wasted. This shows that traditional aggregation algorithms are not efficient in large domains.

[5] `http://www.cs.washington.edu/research/networking/rocketfuel/`

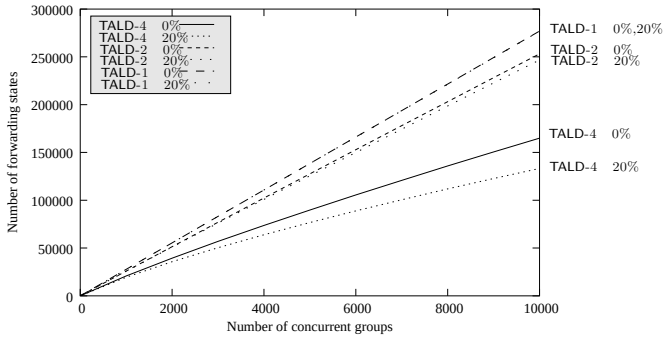

Fig. 3. Number of forwarding states

3.2 Group specific entries

Figure 4 plots the number of group specific entries which are stored in the group-label table and which match groups to the labels of the aggregated trees. As this number is related to the number of groups, it is not dependent of the bandwidth thresholds and the results are equivalent for 0% and for 20% of bandwidth wasted. The protocols TALD-2 and TALD-4 need to store more group specific entries in order to route the packets for the groups between the sub-domains. These entries are stored in order to configure the tunnels crossing the sub-domains. Consequently, TALD-1 does not store such entries.

The results show that TALD-4 needs to store more entries than TALD-2 which in turn stores more entries than TALD-1. This is the price to be paid to achieve aggregation and to reduce the number of forwarding states. Note that the more sub-domains, the larger the number of groups specific entries. Consequently, it may not be interesting to divide the domain into too many sub-domains because the reduction of forwarding states will not be so significant.

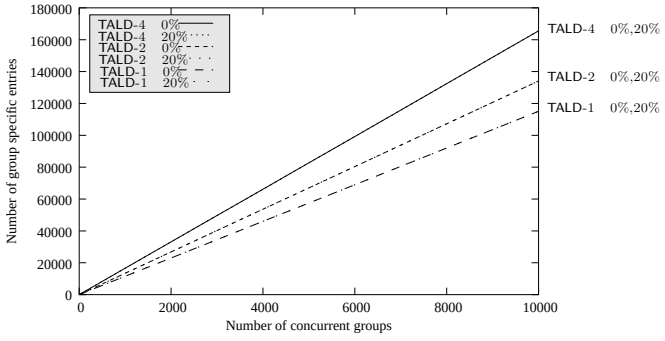

Fig. 4. Group specific entries

However, TALD-4 reduces the total number of entries stored in routers compared to TALD-1. Figure 5 shows the total number of the groups specific entries and the forwarding states stored in all the routers of the domain. TALD-4 achieves a reduction of 16% of this total number compared to TALD-1 when no bandwidth is wasted and a reduction of 25% with 20% of bandwidth wasted. It may be noted that TALD-2 does not achieved significant reduction of this number compared to TALD-1. Consequently, dividing the domain in two sub-domains is not enough. However, the memory in routers is significantly reduced with TALD-4. As the number of group specific entries increases with the number of sub-domains, it is not be interesting to divide more the domain. Indeed, the more the domain is divided, the less number of forwarding states but the more the number of group specific entries.

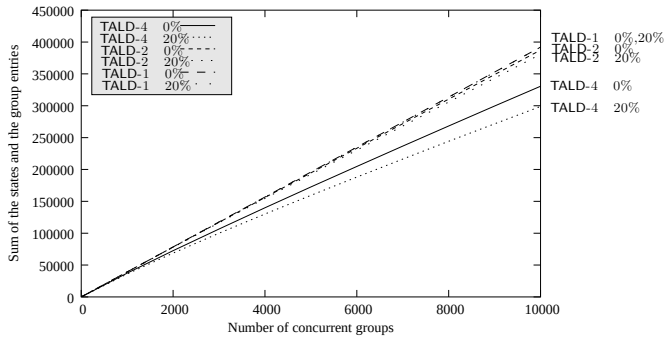

Fig. 5. Sum of the forwarding states and of the group specific entries.

3.3 Aggregation Ratio

Figure 6 shows the aggregation ratio in function of the number of concurrent groups. The aggregation ratio is denoted by the number of trees with aggregation out of the number of trees if no aggregation is performed. Note that for TALD-2 and TALD-4, the number of trees is the sum of the number of trees for each sub-domain.

The protocol TALD-1 achieves less than 1% of aggregation even when 20% of bandwidth is allowed to be wasted. The protocol TALD-4 achieves more than 40% of aggregation even when no bandwidth is allowed to be wasted. When 20% of bandwidth is wasted, the aggregation ratio reaches more than 55%. This figure shows that with large networks, existing algorithms achieving tree aggregation without any division of the domain (as TALD-1) do not realize any aggregation at all.

Figure 7 plots the aggregation ratio in function of the number of border routers in the domain when there are 10 000 concurrent groups. We vary the

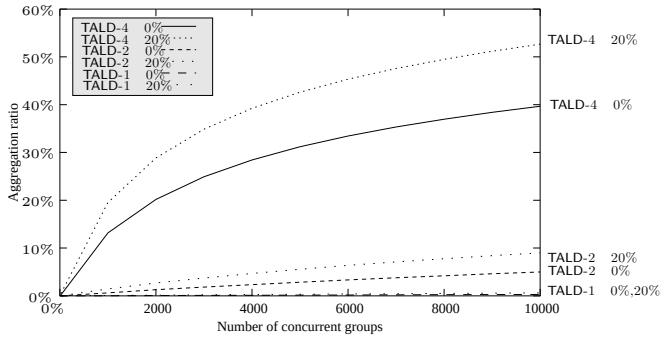

Fig. 6. Aggregation ratio

number of possible border routers among all the 201 routers of Exodus network from 10 to 200. We run 100 times the algorithm for each possible value of the number of border routers in order to get different sets of border routers. The routers that were not border routers could not be attached to members of multicast groups.

With domains of 10 border routers, the aggregation is very efficient and after 10 000 concurrent groups, the protocols are able to aggregate any new group in the domain. The aggregation ratio decreases dramatically, especially for TALD-1 which is not able to perform any aggregation when the domain contains more than 40 border routers. However, TALD-4 is efficient and performs more than 20% of aggregation even when there are 200 border routers. This shows that for a domain of 40 border routers or more, it is strongly recommended to divide the domain into several sub-domains in order to aggregate groups.

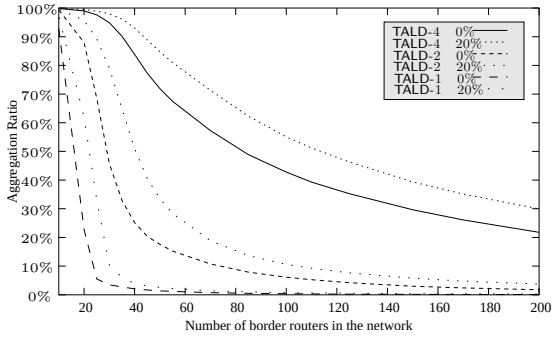

Fig. 7. Aggregation ratio in function of the number of border routers

4 Related Work

We presented in this paper, a tree aggregation protocol specific to large domains. In this section, we give an overview of the protocols achieving tree aggregation already in the literature. Tree aggregation idea was first proposed in [5] and since, several propositions have been written.

The protocol AM [2,3] performs aggregation using a centralized entity called the *tree manager* responsible of assigning labels to groups. The protocol STA [6] proposes to speed up the aggregation algorithm with a fast selection function and an efficient sorting of the trees. These two protocols are represented by TALD-1 during the simulations. TOMA [7] is a recent protocol that performs tree aggregation in overlay networks.

Distributed tree aggregation. The distributed protocol BEAM proposed in [4] configures several routers to take in charge the aggregation in order to ditribute the work load of the *tree manager*. Indeed, in AM or in STA, only the *tree manager* takes this responsibility. The protocol DMTA [9] proposes to distribute the task of the tree manager among the border routers and then to suppress completely the requests to centralized entities necessary in BEAM to achieve aggregation.

Tree aggregation with bandwidth constraints. AQoSM [1] and Q-STA [10] achieve tree aggregation in case of bandwidth constraints. In these two algorithms, links have limited bandwidth capacities and the groups have different bandwidth requirements. Consequently, groups may be refused if no tree can be built satisfying the bandwidth requirements. Q-STA accepts more groups as this protocol builds native tree maximizing the bandwidth available on the links.

Tree aggregation with tree splitting. The protocol AMBTS [8] performs tree splitting before aggregating groups in order to manage larger domains. A tree is divided into several sub-trees and whenever a new group arrives the native tree is splitted in sub-trees according to a foreclosing process. From these sub-trees, the *tree manager* tries to find already existing sub-trees and to aggregate the group. The idea of AMBTS is somehow orthogonal to the idea of TALD. However, we did not compare AMBTS to TALD during the simulations because of the following reasons.

First of all, the protocol is not realistic for large domains as a centralized entity is responsible of all the process of aggregation. This centralized entity keeps the group memberships for all the groups of the whole domain. Moreover, it is in charge of splitting the trees and aggregating the groups. This behavior is not scalable in domains such as Exodus network with 200 routers. Indeed, too much memory is used to store all the information and the centralized entity is strongly solicited each time a member of a group changes. Second, the foreclosing process in which a tree is divided into several sub-trees is not detailed and we were not able to simulate this algorithm due to lack of information. Splitting the trees manually was not possible in our domain. Finally, the number of sub-trees grows tremendously and is larger than the number of groups (especially if the trees are splitted in many sub-trees). Thus, the process of aggregation is strongly

slowed down due to the large number of evaluations of sub-trees. In AMBTS, the simulations were done on a network with 16 border routers. All these reasons make us decide to propose and detail a protocol adequate to large domains.

5 Conclusion

In this paper, we proposed a protocol that achieves aggregation in large domains. Indeed, previous known algorithms were not able to perform any aggregation in this case. Consequently, current tree aggregation protocol were not able to reduce the number of forwarding states and behaved in the same way as traditional multicast. The main idea of our protocol is to divide the domain in several sub-domains and to aggregate the groups in each sub-domain. The simulations showed that in large domains where no aggregation was performed, our protocol behaves well and gives good results. The aggregation ratio was around 20% for a domain with 200 border routers while actual protocols achieved 0% of aggregation for domains of more than 40 border routers.

This work leads to many perspectives of research. First, the connection of the trees in each of the sub-domain can be achieved in different ways. Presently, it is done by configuring tunnels however, this connection can be achieved by a tree for example. Second, the domain can be divided using an adaptive algorithm and in more sub-domains, thus it may be interesting to study the impact of this division on the aggregation.

References

1. J.-H. Cui, J. Kim, A. Fei, M. Faloutsos, and M. Gerla. Scalable QoS Multicast Provisioning in Diff-Serv-Supported MPLS Networks. In *IEEE Globecom*, 2002.
2. J.-H. Cui, J. Kim, D. Maggiorini, K. Boussetta, and M. Gerla. Aggregated multicast, a comparative study. In *IFIP Networking*, number 2497 in LNCS, 2002.
3. J.-H. Cui, J. Kim, D. Maggiorini, K. Boussetta, and M. Gerla. Aggregated Multicast — A Comparative Study. *Special issue of Cluster Computing: The Journal of Networks, Software and Applications*, 2003.
4. J.-H. Cui, L. Lao, D. Maggiorini, and M. Gerla. BEAM: A Distributed Aggregated Multicast Protocol Using Bi-directional Trees. In *IEEE International Conference on Communications (ICC)*, May 2003.
5. M. Gerla, A. Fei, J.-H. Cui, and M. Faloutsos. Aggregated Multicast for Scalable QoS Multicast Provisioning. In *Tyrrhenian International Workshop on Digital Communications*, September 2001.
6. A. Guitton and J. Moulierac. Scalable Tree Aggregation for Multicast. In *8th International Conference on Telecommunications (ConTEL)*, June 2005.
7. L. Lao, J.-H. Cui, and M. Gerla. TOMA: A Viable Solution for Large-Scale Multicast Service Support. In *IFIP Networking*, number 3462 in LNCS, May 2005.
8. Z.-F. Liu, W.-H. Dou, and Y.-J. Liu. AMBTS: A Scheme of Aggregated Multicast Based on Tree Splitting. In *IFIP Networking*, number 3042 in LNCS, 2004.
9. J. Moulierac and A. Guitton. Distributed Multicast Tree Aggregation. Technical Report 5636, INRIA, July 2005.
10. J. Moulierac and A. Guitton. QoS Scalable Tree Aggregation. In *IFIP Networking*, number 3462 in LNCS, 2005.